Definition, modeling and simulation of a grid computing system for high throughput computing

Eddy Caron, LIP/ENS Lyon, 46 Alle d'Italie, 69364 Lyon Cedex 07
V. Garonne, A. Tsaregorodtsev
Centre de Physique des Particules de Marseille, Marseille, France

January 29, 2006

Public Note

Issue : 1
Revision : 0
Reference : LHCb-2005-083-Offline
Created : January 12, 2005
Last modified : January 29, 2006

Abstract

In this paper, we study and compare grid and global computing systems and outline the benefits of having an hybrid system called *DIRAC*. To evaluate the *DIRAC* scheduling for high throughput computing, a new model is presented and a simulator was developed for many clusters of heterogeneous nodes belonging to a local network. These clusters are assumed to be connected to each other through a global network and each cluster is managed via a local scheduler which is shared by many users. We validate our simulator by comparing the experimental and analytical results of a M/M/4 queuing system. Next, we do the comparison with a real batch system and we obtain an average error of 10.5% for the response time and 12% for the makespan. We conclude that the simulator is realistic and well describes the behaviour of a large-scale system. Thus we can study the scheduling of our system called *DIRAC* in a high throughput context. We justify our decentralized, adaptive and opportunistic approach in comparison to a centralized approach in such a context.

0.1 Introduction

High energy physics, particularly the CERN LHCb experiment, need a lot of computing resources to verify aspects of the detector design and algorithms. These tasks required a large number of *parameter sweep* (1) simulations which strongly favors *high-throughput computing* (2). This is typical of situations where the supply of computational jobs greatly exceeds the available resources and jobs are generally not time critical. So it is not surprising that its applications have been evolving with an institutional, large-scale system environment in mind.

In these systems, resources are heterogeneous clusters, each accessible from a local network and belonging to a single administrative domain. To aggregate theses clusters and manage the workload a global architecture must be defined paying special attention to the size of these systems, *e.g.* the envisaged size is around a hundred sites spread all over the world containing 30,000 nodes. While batch systems are often used at the local level, there is no common solution in a global context. In the paper, we explain how the *DIRAC* (Distributed Infrastructure with Remote Agent Control) system has been conceived and developed by CERN LHCb experiment to meet these requirements and provide a generic, robust grid computing environment. The paper is organized as follows:

Section 0.2 presents the background and the *DIRAC* proposal for the meta-scheduling issues; Section 0.3 presents the model used; Section 0.4 discusses the simulation tool ; Section 0.5 describes how it was validated; Section 0.6 shows the experimental setup; Section 0.7 shows the experimental results and finally Section 0.9 finishes with conclusion and future plans.

0.2 Background: Grid vs. Global Computing

We categorize large-scale, distributed computational systems into two groups (3): grid computing and global computing systems. Grid computing uses common interfaces to link together computing clusters. The aim is to allow Virtual Organizations of users who span institutional boundaries to share computing resources, usually clusters. These clusters are shared between many users or virtual organizations (VO) (4) and a local policy is applied to each cluster which defines their access rights. This policy is applied through a resource management system, *i.e.* a batch system.

Table 1: Comparison of the principal features of grid and global computing systems.

Attributes	**Grid Computing**	**Global Computing**
View of conception	administrator	*user*
System size	100 sites, 10000 nodes	*1 million nodes*
Type of computing resource	*clusters*	PCs
Platform	*stable*	volatile
System deployment	intrusive	*light*
Software distribution	manually by site managers	*para-trooper, on the fly*
Application type	*multiple, generic*	dedicated
Job length	*long*	*short*
Direct submission	*yes*	no
User organization	severals communities(VOs)	*one community*
Security	*important*	weak
Authentification	*identification*	anonymous
Scheduling paradigm	Push, sender-initiated	Pull, resource-initiated

Global computing is an *Ad hoc* network of individual computers, *e.g.* desktop machines, which act as slaves for a central job server which usually supports a single *parameter sweep application.* Table 1 compares the principal features of these two systems.

In a multi-site grid project (5; 6; 7), scheduling decisions are often taken with a global view of the system. The architecture in Figure 1 is composed of a centralized meta-scheduler and an information system.

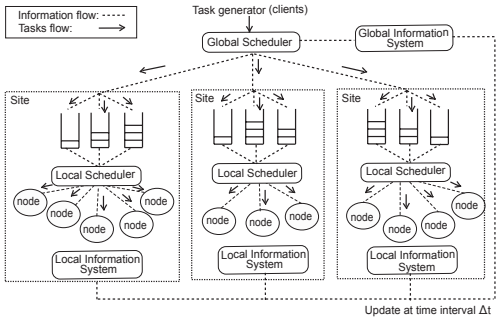

Figure 1: Example of an architecture with centralized scheduling

This approach uses the "Push" paradigm. In this model, the global information system, *e.g.* MDS (8), keeps all the static and dynamic information about the system state. Sensors deployed on the sites update the information by first querying the local information system and then updating their own information in the global information system. Ideally these updates occur for each state change, for example, the arrival or end of a task. In fact, this solution often generates a message storm and needs some kind notification mechanism. The use of a finite, update period $\triangle t$ seems most appropriate and stems the flow of messages. So far no grid project is able to manage the workload on the order of 100 sites. A global computing system uses instead the "pull" scheduling paradigm. These operate in a cycle-scavenging mode, using idle CPU power and pulling jobs from a central server. Some examples of this include SETI@Home, BOINC, and distributed.net. This paradigm is only applied to restricted resources such as simple PCs. Some systems can connected thousands of nodes working concurrently.

Grid infrastructures are heavier, more intrusive and less flexible than global computing systems which are lightweight, flexible and dynamic. However, global computing systems serve limited applications in a non-secure environment. By looking at the complementary features of these systems, we see today a convergence between these two systems on what we called virtualization, *e.g. Service Oriented Architecture*. We italisized in Table 1 the relevant characteristics for an hybrid system which incorporates aspects of grid, global computing paradigms. The *DIRAC* system is such an improved hybrid.

DIRAC is organized with a number of independent services to respond the needs of one VO cause as we observed, a single system for one VO is more scalable as the total member of a VO stay stable and do not vary a lot in time. Such an infrastructure should deals with the virtualization of resources in order to make the usage of heterogeneous resources transparent, to globalize and aggregate all the resources available for one particular VO.

Within the *DIRAC* resource management service we try to provide a multi-site system for high throughput computing. We propose a scheduling model, illustrated in Figure 2, which deploys agents on the sites and uses central global queues. Using the cycle-stealing paradigm borrowed from global computing, it uses a *'Pull'* paradigm where agents demand a task if they detect a free slot. *DIRAC* extends this concept to different computing resources by defining a criterion of availability. These resources could be anything from a simple PC to whole batch systems.

As soon as a resource is detected to be available the dedicated agent requests tasks from the match-maker service, which contains the dynamic and static information about the resource.

The most important issue is knowing which paradigm is more appropriate for high throughput computing. Some studies (9) propose strategies which employ file queuing systems (10) whilst others (11) use simulation mechanisms like BRICKS (12). Generally these work quite well but only in a simplified and unrealistic model. We propose in this paper a realistic model and a simulator to evaluate a scheduling paradigm in a multi-site context.

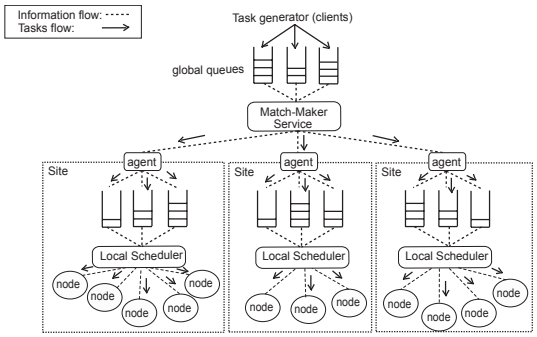

Figure 2: The *DIRAC* scheduling model.

0.3 The performance model

Let $\mathcal{C}$ represent the set of clusters present in the multi-site platform. Each cluster $\mathcal{C}_i$ owns a set of worker nodes $\mathcal{N}_i$ and belongs to a local domain, i.e. a LAN (Local Area Network). This local network describes a graph for the nodes. Each link of this graph has a **local bandwidth** $bwt_{\mathcal{C}_i}$ and a **local latency** $latency_{\mathcal{C}_i}$.

A cluster $\mathcal{C}_i$ is connected to the global network or WAN (World Area Network) by a switch. Let $bwt_{\mathcal{C}}$ be the **global bandwidth** and $latency_{\mathcal{C}}$ the **global latency**. Many different approaches exist to generate the graph for the proposed model and some recent studies show that networks follow specific power laws (13).

Let (i, j) be the pair defines the j^{th} node of cluster $\mathcal{C}_i$. Each node (i, j) has a processor capacity $capacity_{i,j}$ and to express this we define one computing unit, the ***NCU*** (Normalized Computing Unit). This unit is determined by special application benchmarking on different referential machines, taking into account the absolute time. So the capacity of a node is simply the total number of computing units able to be computed per unit time. We can then model the platform heterogeneity.

0.3.1 The realistic workload model

We define two levels for the workload: local and global. The global workload corresponds to the tasks submitted to the metacomputing system, usually called meta-tasks. The local or background workload corresponds to tasks locally submitted to a cluster. A meta-task m_k is mapped locally to a task k.

A task k has four attributes : $attributes_k$=$\{tl_k, length_k, proc_k, group_k\}$ where tl_k is the **local submission date**, $length_k$ the **length** expressed in NCU, $proc_k$ is the **total number of processors** required for the task execution and $group_k$ the **organization** who submits the task. A meta-task m_k is composed of the task properties sub-set and the **global submission date** t_k. So we have $meta\text{-}attributes_k$=$\{t_k, attributes_k\}$.

Modelling the workload for a metacomputing system involves determining k for each task from the task set $\mathcal{T}$ then submitting the attributes $attributes_k$ to a cluster C_i. The meta-tasks m_k of the set of meta-tasks $\mathcal{MT}$ and their meta-attributes are also submitted to the system. Possible methods for generating a workload are the following: a randomized workload, a workload derived from real system traces and lastly a stochastic workload. A random workload and a workload derived from a real system are not realistic. A workload derived is judged to represent too many platform dependant characteristic and so too specific. Instead, the stochastic workload is chosen here. Some works (14) studying computing centre traces propose a complete probabilistic model and so we write $S(\mathcal{T})$ for the **distribution function which generates the length set** for a set of tasks $\mathcal{T}$. Let CA be the **cut applied** to this length set which fixes the maximal and minimal length. We also denote the **distribution function which generates the submission date set** by $A(\mathcal{T})$ for a set task and finally the **average submission rate** by $\lambda_{\mathcal{T}}$.

0.3.2 The local model scheduling

At the local level, nodes of a same site are typically managed by a resource management system, e.g a batch system. Other implementations use queues which are defined by the characteristics of the tasks, for example, their length. Shared time scheduling between users is done by the local scheduler which would normally apply policies based on quotas or priorities. For a cluster $\mathcal{C}_i$, we can define a **queues set** $\mathcal{Q}_i$. Each queue $\mathbf{q_{i,j}}$ of $\mathcal{Q}_i$ is composed of a set of nodes $\mathbf{N_{q_{i,j}}}$. Any particular node can belong to one or many queues. The tasks submitted to the site are then added to these queues to wait for their execution. Subsequently, a queue $\mathbf{q_{i,j}}$ will contain a task set $\mathcal{T}_{i,l}$. We define the **queue depth** $depth_{i,j} = card(\mathcal{T}_{i,l})$ as the **total number of tasks waiting** in the queue a particular instant. The **maximal time** that a task could spend in execution on a node of the queue $q_{i,j}$ is denoted by $t_{max_{i,j}}$.

0.3.3 Measures and metrics

For a task k, we define three following states: *queued*, *running* and *done*. The state *queued* means that a task is in a waiting queue. When the task is executing it is in the *running* state and the *done* state signals that the task is completed. The corresponding times for the changing states *running, queued* and *done* for a task k are respectively r_k, q_k and d_k. The **local waiting time for a task** k is the execution beginning time minus the submission time $r_k - tl_k$. The **execution time** is $d_k - r_k$ and the **local response time** is $d_k - tl_k$. For a meta-task mk, we have a **global waiting time** which is the beginning execution time minus the global submission time, $r_k - t_k$. The **global response time** is $d_k - t_k$. A set of derived metrics based on these measurements appears in Table 2. The **makespan** is the full time to complete all the jobs.

Table 2: Metrics defined for the set of meta-tasks $\mathcal{MT}$.

Name	Notation
Average waiting time	$waiting_m$
Average execution time	$execution_m$
Average response time	$response_m$
Makespan	$\max_{k \in \mathcal{MT}}(d_k) - \min_{k \in \mathcal{MT}}(t_k)$

0.4 Simulation tool

Simgrid (15) is a discrete event toolbox which allows the modeling and description of a platform for centralized, hierarchical or fully distributed scheduling. *Simgrid* with the following improvements was used for this study:

Description platform module. Our simulator is interfaced with the hierarchical graph generator *Tiers* (16). For the capacity information, we define a sample set of nodes where each node is weighted by a percentage. This percentage expresses the proportion of this node type present in the platform. The node NCU capacities and their weights are inspired by the performances obtained by *DIRAC* for a physics application on the production platform. This platform was composed of more than 4,000 nodes and twenty different node configurations. Based on the total number of nodes and their proportions we generate the set $\mathcal{P}$ of all available capacity. Then for each node we proceed by drawing lots in the set $\mathcal{P}$ and one occurrence of this value would be removed from the set $\mathcal{P}$ until the capacity attribute is filled for all nodes.

Workload generator. *Simgrid* has already got an implementation of the task concept. In this model, however, the meta-data is added. The workload generator provides different probability density functions like the Gamma law, Gaussian law and so on. To have a shared system we implement an agent per client or organization. The simulation tool allows one to simulate different system users. Therefore, it is possible to have different workloads submitted at the same time and evaluate their interactions.

Generic batch system. The basic entity at the local level is the batch system. As *Simgrid* does not provide a model for this, a generic one has been implemented. The design is illustrated in Figure 3.

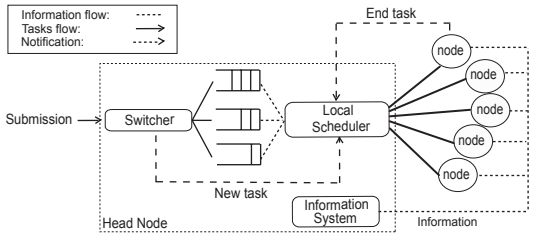

Figure 3: The generic Batch system design.

A head node hosts the principal components: switcher, queues, information system and finally the local scheduler. Each node communicates with the head node. A task submission is managed by the switcher which with regard the task requirements, places it in a queue and notifies the scheduler. This then queries the monitoring and accounting system to choose a candidate node. If no resource is available the task stays in the queue but once the task is sent to the node it is executed. After the task is completed, the scheduler is notified which triggers a cycle where the scheduler looks at the queues and determines if another task could be executed. The scheduler configuration is entered by file and includes properties such as the total number of queues, the availability or not of a node in a queue and the maximum number of tasks which could execute on a node.

Meta-scheduling architectures. Two kinds of global architectures were implemented. First the centralized architecture and second the *DIRAC* architecture outlined in Section 0.2.

0.5 Validation of the simulation tool

For the theoretical validation, experiments were performed on M/M/m queuing systems (10). Figure 4 shows the response time differences between the simulated results and the analytical theory for a M/M/4 system. The service time average is four units time. The arrival rate follows an exponential law. The simulated responses are derived from 16 independent runs of 1,000 tasks and the root mean square error for all simulated arrival rates. The results obtained are consistent with theory.

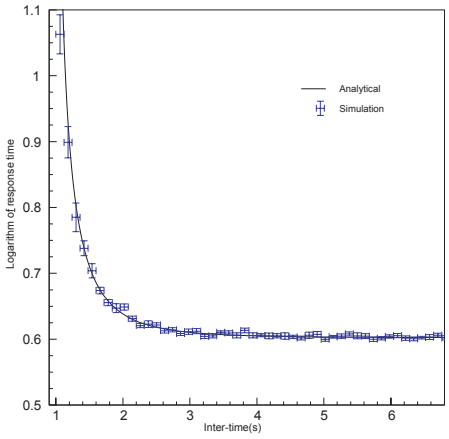

Figure 4: Response time comparison between simulation and M/M/4 system.

0.5.1 Experimental Comparison

A dedicated and heterogeneous cluster was used, described in Table 3. A *DIRAC* agent was deployed on the cluster with a task generator put in place. This generator submits sequential tasks which have no dependencies or communication between them. The submission times follow a Poisson law and the

benchmark used was a program which implements a CPU consumed counter. It takes one parameter which is the number of CPU to consume before ending. This length is created for each task and follows a Weibull law. The response time and the waiting time are then captured by the *DIRAC* monitoring service. Then, we capture this workload to inject it in our simulator. To estimate the execution time according to the node capacity we normalize this time with the node NCU capacity. The NCU node capacity is determined by benchmarking, outlined in Table 3. A simple topology is assumed where each node is connected to the head node by a simple link with 100 megabyte/s bandwidth and no latency, as illustrated in Figure 3.

Table 3: Platform caracteristics.

Attributes	**Values**		
Total number of nodes	3		
Type	PII	PII	PIII
Processor (MHz)	350	400	600
Memory (mo)	128	128	128
Capacity ($NCU.s^{-1}$)	32.12	52.12	100.00
Scheduler	openPBSv2.3		
Politicy	FCFS		
Local Network	Megabyte Ethernet		

The total number of tasks is 330, i.e. $card(\mathcal{MT}) = 330$. We observe for first instance an average error of 80% for the response time. After a trace study we characterize two service times, μ_{rec} and μ_{send}. μ_{rec} is the service time between task arrival and task sending on a node or in queue and μ_{send} is the necessary time to notify the scheduler of a task completion. This large average error can be explained by the fact that the scheduler made its choice with a different system state view. The nodes are heterogeneous so the consequences are dramatic for the response time. We correct this error by including the service times μ_{rec} and μ_{send} measured on the real system injected as traces. Then, we obtain exactly what we would expect in reality which validates the code. The experiment is then repeated by setting time services to constants. These constants are the average service time observed in reality ($\mu_{rec} = 3.75s$, $\mu_{send} = 2s$). After this we observe an improvement in the average error of 10.5%.

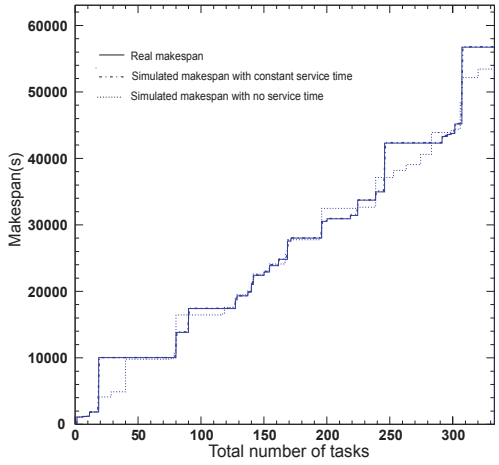

Figure 5: Comparison between simulation and a real batch system with $\mu_{rec} = \mu_{send} = 0$ and $\mu_{rec} = constant$, $\mu_{env} = constant$. Makespan evolution vs. the number of tasks.

Figure 5 shows the makespan evolution versus the total number of tasks. For the constant service time, we see an average error of 12%. From this it is possible to conclude that at the local level the simulator is realistic. It is now possible to proceed to the strategies and meta-scheduling architectures evaluation. One further improvement could be to make the service times μ_{rec} and μ_{send} a distribution function approaching the real behavior.

0.6 Experimental setup

The decentralized *DIRAC* architecture and the centralized approach from Section 0.2 can now be compared. The message control size for both architectures is 30 KB in the simulation. The workload characteristics are inspired by an empirical study (14) and the table 0.10 in Appendix summarises the platform parameters and workload attributes.

The associated *DIRAC* strategy is detailed in Section 0.2. The criterion of availability is expressed as $\frac{depth_{i,j}}{card(\mathcal{N}_{q_{i,j}})} < \varepsilon,\ e.g.\ \varepsilon = 0.3$ which implies that tasks in the waiting state scheduled on a computing resource should not exceed 30% of the total number of nodes.

The policy applied at the match-maker level is that of FRFS (Fit Resource First Serve). That means that the first resource which matches well is chosen.

We also propose to evaluate the impact of the deployment in *DIRAC*. Let us consider two kinds of agent deployment. The static approach is described in Section 0.2 whereas the dynamic approach is a concept similar to the resource reservation. After detecting the availability, the agent deployed on the site queries the match-maker to ask if tasks are available. In the case of a positive answer, it submits an agent wrapped in simple task to the cluster. Once the agent arrives at the node, it checks the node capacity and environment and creates the resource description accordingly. After that the agent queries a task from the match-maker. If no task is returned the agent dies. In the simple reservation mode 'Run Once', the agent dies after the completion of the first task while in a 'filling' mode it queries for one more task with respect to the available time.

The algorithm for the centralized scheduling is the following. At each task arrival the scheduler looks for the least loaded resource, *i.e.* the resource $q_{i,j}$ from cluster C_i which has the minimum measured depth $depth_{i,j}$ with $\forall i \in \mathcal{C}$ and $\forall j \in \mathcal{F}_i$.

Now we compare two approaches which strongly favour high throughput computing but the question is: what architectures and implementations could influence their performance?

0.7 Results

Figures 6 show the evolution of total number of tasks in the state *queued* and *running* during the experiment. The third line is the *done* cumulated task curve. The saturation phase gives us maximal capacity of the platform which is equivalent to the sum of all nodes, 60. Both approaches saturate all resources but the evolution is different in the DIRAC approach where the number of tasks in the queued state is constant (Figure 6 (up)).

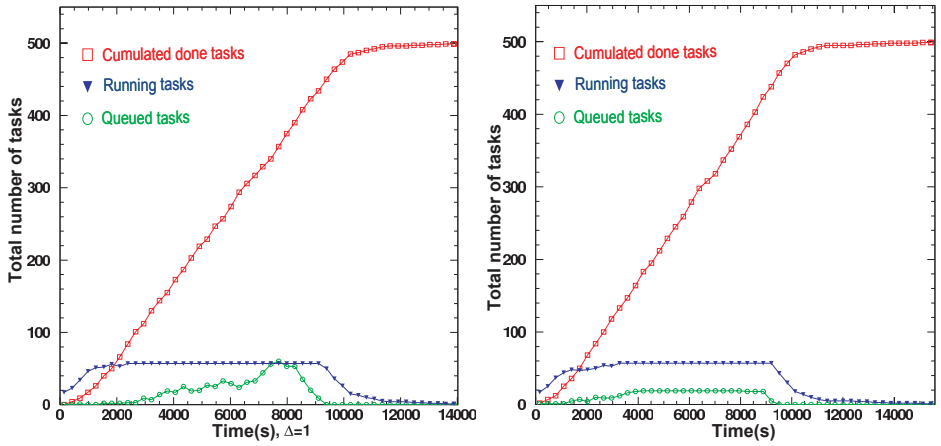

Figure 6: Tasks evolution vs. time in a dedicated platform. Centralized approach with $\triangle t = 0$(up). *DIRAC* approach (down).

Figure 7 shows the variation of the $\triangle t$ period versus average waiting time $waiting_m$ for centralized scheduling in first a dedicated context and then in a shared context. The *DIRAC* waiting times are qualitatively indicated because they are independent in philosophy from $\triangle t$ (Figure 7 (down)).

DIRAC does not use a central information service so does not depend on this period. For a $\triangle t$ less than 95s, the waiting time is better than the centralized scheduling in a dedicated context and performs better by around 60s in the shared context. In the latter the performances rapidly degrade and a more chaotic effect is observed. The upper bound observed corresponds to the situation where all tasks are scheduled on the same site where $\triangle t > \max_{k \in \mathcal{M}t} t_k$.

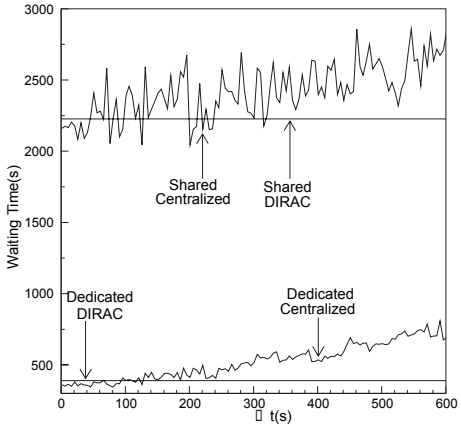

Figure 7: Average waiting time for meta-task vs. period $\triangle$t for the centralized approach in a shared and a dedicated context. Full variation and Comparison with *DIRAC*.

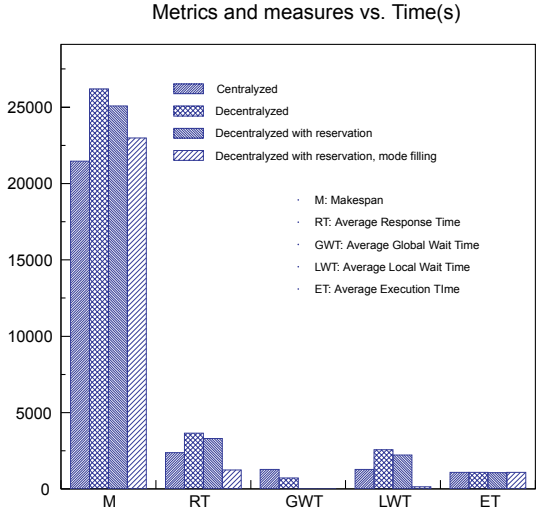

Figure 8: Strategies comparison for: centralized approach, *DIRAC* with a static deployment and the two *DIRAC* reservation mode simple and filling.

Figure 8 compares the makespan as well as the local and global response times executed for the four evaluated strategies. For a null $\triangle t$, the best makespan is obtained for the centralized approach although the smallest response time came from the *DIRAC* approach in the *filling* reservation mode. The execution times are of the same order for all strategies. This is explained by the platform characteristic that sites have the same capacity on average. The response time difference is mainly due to the local and global waiting time. The largest local waiting time is found with the static *DIRAC* scheduling however, the global waiting time in this situation is minimal.

In the case of *DIRAC* reservation the local waiting times are null because the matching is done directly from the node. The waiting time is expressed for the agents in this case. The effect of changing the deploy-

ment from static to dynamic gives a improvement of 10% for the average response time. The reservation mode *filling* nearly introduced a 50% improvement for the average response time in comparison with the centralized approach.

0.8 Discussion

In an ideal situation the centralized approach gives the best results but it is often impractical to assume that such a platform would stay stable. Common failures, by order of importance are: network failure; the disk quotas; unavailability of services; incorrect local configuration and finally power cut. With this large scope of error it is difficult to keep an ideal view of a global system. The scheduling is totally dependant on the information system performance and this system often does not scale well.

DIRAC bypasses this problem because one of the main characteristics is the total absence of a system global view. It takes its decisions with a partial view. Each resource, in conjunction with its current state, gets an appropriate workload to suit its capacity. The tasks are put in a buffer where the scheduling event is an attribute of the resource availability which is the opposite to the centralized approach where the triggered event is a task submission.

If a platform deterioration occurs, any drawback from using the centralized approach is immediately paid back in term of performance. This effect is also more significant if the approach is combined with predictions. A rapid state change of a resource is taken into account only after a lapse of time in the centralized model. During this lapse in a high throughput context, the decisions made can be disastrous. Resource starvations and system information failures are also two main drawbacks which do not affect *DIRAC*, where all available resources are utilized immediately.

DIRAC demonstrates adaptability. This dynamic aspect forces scheduling in an opportunistic, reactive and non-predictive way. On the other hand, the results are quite similar with the centralized scheduling. *DIRAC* is easy to implement, stable and flexible. It also facilitates resource reservation which can significantly increase performance. Nevertheless it must be said that technically this improvement required direct communication with worker nodes. Within *DIRAC* passive communication mode, relaying of outbound connectivity is used to accomplish this.

The reservation mode causes a higher and more regulated load on the match-maker service. This penalty for this improvement is the huge number of agents which abort right after the non-task answer (299 in our case which is non-negligible).

0.9 Conclusion and future work

In this paper we propose a model for a meta-scheduling platform. We measure an average error of 12% for the makespan prediction. With this tool it is demonstrated that a centralized approach is better than a decentralized approach in term of performances for high throughput computing. However, this happens only in the ideal case where the update period is quasi null. Above 95s in a dedicated context, the 'pull' approach had similar results and importantly was more stable. The same observation is made in a shared context. The 'pull' approach also provides an abundance of scenarios which allow a performance enhancement of just under 50% compared to the centralized approach. This was most evident with resource reservation. It will be interesting to study the impact of the migration from site to site with regard to their local workload. Future work will be into the study of this aspect.

Acknowledgment

We gratefully acknowledge Charles Loomis and Stuart Paterson for helping to prepare this document.

Bibliography

[1] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, Heuristics for scheduling parameter sweep applications in grid environments, in: Heterogeneous Computing Workshop, 2000, pp. 349–363.

[2] M. Livny, J. Basney, R. Raman, T. Tannenbaum, Mechanisms for high throughput computing, SPEEDUP Journal 11 (1).

[3] I. T. Foster, A. Iamnitchi, On death, taxes, and the convergence of peer-to-peer and grid computing., in: Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers, 2003, pp. 118–128.

[4] I. Foster, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, Lecture Notes in Computer Science 2150.

[5] S. Vadhiyar, J. Dongarra, A metascheduler for the grid, in: Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing, 2002.

[6] V. Hamscher, U. Schwiegelshohn, A. Streit, R. Yahyapour, Evaluation of job-scheduling strategies for grid computing, in: Proceedings of the First IEEE/ACM International Workshop on Grid Computing, Springer-Verlag, 2000, pp. 191–202.

[7] A. Bose, B. Wickman, C. Wood, Mars: A metascheduler for distributed resources in campus grids, in: GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), IEEE Computer Society, Washington, DC, USA, 2004, pp. 110–118.

[8] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, A directory service for configuring high-performance distributed computations, in: Proc. 6th IEEE Symp. on High Performance Distributed Computing, IEEE Computer Society Press, 1997, pp. 365–375.

[9] L. He, S. A. Jarvis, D. P. Spooner, G. R. Nudd, Optimising static workload allocation in multiclusters., in: IPDPS, 2004.

[10] L. Kleinrock (Ed.), Queueing Systems Volume I : Theory, John Wiley and Sons, 1975.

[11] A. Takefusa, S. Matsuoka, H. Casanova, F. Berman, A study of deadline scheduling for client-server systems on the computational grid, in: HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), IEEE Computer Society, 2001, p. 406.

[12] A. Takefusa, Bricks: A performance evaluation system for scheduling algorithms on the grids., in: JSPS Workshop on Applied Information Technology for Science, 2001.

[13] D. Lu, P. Dinda, Synthesizing realistic computational grids, in: Proceedings of ACM/IEEE SC 2003, 2003.

[14] H. Li, D. Groep, L. Walters, Workload characteristics of a multi-cluster supercomputer, in: Job Scheduling Strategies for Parallel Processing, Springer-Verlag, 2004.

[15] H. Casanova, Simgrid: A toolkit for the simulation of application scheduling, in: Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), Brisbane, Australia, 2001.

[16] Doar, A better model foor generating test networks, in: IEEE GLOBECOM, 1996.

0.10 APPENDIX - Experiments parameters

	Parameters	Notations	Values
Platform	Total number of sites	$card(C)$	3
	Total number of node per site	$card(N_i)$	20
	Total number of queue per site	$card(F_i)$	1
	mean node capacity	$capacity_m$	96 $NCU.s^{-1}$
	Local policy	M/M/$card(N_i)$/FCFS	FCFS
	Maximal execution time	$t_{max_{i,j}}$	24000s
	local/global bandwidth	bwt_C/bwt_{C_i}	1000 $Mbit$/100 $Mbit$
	local/global latency	$latency_C, latency_{C_i}$	0s
Workload	Task Type	$card(proc_k)$	1
	Length distribution	$S(\mathcal{M}t) \rightarrow \{size_k\}$	$Weibull(\alpha = 142.2, \beta = 0.45)$
	Length cut	$\mathcal{C}(length_k)$	37300 $< length_k <$ 242800
Global	Total number of task	$card(\mathcal{M}t)$	500
	Arrival time distribution	$A(\mathcal{M}t) \rightarrow \{tl_k\}$	$Poisson(m = 0.05, s = 4)$
	mean inter-arrival	$1/\lambda_{\mathcal{M}t}$	19s
Local	Total number of task per site	$card(\mathcal{T}_i)$	500
	Arrival time distribution	$A(\mathcal{T}_i) \rightarrow \{t_k\}$	$Poisson(m = 0.011, s = 4)$
	mean inter-arrival	$1/\lambda_{\mathcal{T}_i}$	87s